Attorney's Docket No.: 10005265-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : MAMOUN ABU-SAMAHA Art Unit : 2654
Serial No. : 09/684,065 Examiner : Lerner, Martin
Filed : October 6, 2000
Title : DISTRIBUTED VOICE AND WIRELESS INTERFACE MODULES FOR EXPOSING MESSAGING/COLLABORATION DATA TO VOICE AND WIRELESS DEVICES

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 CFR § 1.132

I, Stanley Foster, hereby declare as follows.

1. I am a technical contributor at Hewlett-Packard Company, which is the assignee of the present patent application identified above (referred to herein as the "present patent application).

2. In my capacity as a technical contributor, I work on designing, developing and deploying Microsoft Windows solutions, at least some of which integrate with Microsoft Exchange Server.

3. I have been a software engineer for over 30 years. I am certified in Microsoft Windows programming including .Net and XML. During the last 10 years my work has focused primarily on developing and deploying software solutions involving Microsoft Windows and Microsoft Exchange.

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450 on:

June 12, 2006
Date

(Signature of person mailing papers)

Edouard Garcia
(Typed or printed name of person mailing papers)

4. Paragraphs 10 and 13 of this Declaration refer to BEN GOETTER, DEVELOPING APPLICATIONS FOR MICROSOFT® EXCHANGE WITH C++, 14-15 (1996), which is attached hereto as Exhibit A.

5. I have read the present patent application, including the pending claims.

6. On page 16, lines 16-19, the present patent application discloses that the access module 120 manages sessions by creating XML structures for holding long variable names and references and by passing only simple references to the destination device. When the destination device needs particular data, the simple reference may be replaced on the server with the actual referenced data, which may be passed to the messaging/collaboration server.

7. On page 18, lines 10-34, the present patent application discloses a sessionID_Inbox.xml file that contains "ID=" statements, which associate respective simple references (i.e., the ID type values "1", "2", "3", "4", and "5") with GUID (Globally Unique IDentifier) numbers that reference respective messages in an "Inbox" folder that is maintained by a Microsoft Exchange server on which the access module 120 is executing.

8. Both before and since October 6, 2000, various versions of the Microsoft Exchange Server software have used unique, global pseudo-random 128-bit GUID numbers to identify items maintained in messaging/collaboration folders (e.g., contacts, calendar, e-mail and tasks folders).

9. Both before and since October 6, 2000, application programs routinely have retrieved items from Microsoft Exchange Server folders by sending the server the GUIDs for the folders and the items to be retrieved.

10. For a brief introduction to the use of GUIDs by Microsoft Exchange Server applications, see for example, Exhibit A, GOETTER, *supra* paragraph 4, at 14, which explains that:

> For a client to ask a component object for an interface, the client needs to have a name for that interface: a token that the client and

> component agree represents the interface and that neither can mistake for any other interface. Recall that any change to an interface results in a new interface distinct from the old one.
>
> Every named object in the universe of COM has as its true name a Globally Unique Identifier, or *GUID*. A GUID is a 128-bit integer, spanning a theoretical range from 0 through (2^{128})-1, or approximately 3.4×10^{38}.

11. The number following the folder label "Inbox" and contained within quotes in sessionID_Inbox.xml file on page 18, lines 11-14, of the present patent application is the GUID for the "Inbox" folder that is maintained by the Microsoft Exchange server on which the access module 120 is executing.

12. Similarly, the 128-bit numbers following the brackets ">" in the ID="[number]" statements in the sessionID_Inbox.xml file on page 18, lines 10-34 of the present patent application are the GUID reference numbers that are used by the Microsoft Exchange server to identify messages in the "Inbox" folder.

13. For an example of a conventional way in which a GUID may be presented, see for example, Exhibit A, GOETTER, *supra* paragraph 4, at 15, which explains that:

> By convention, we write a GUID value as a series of hexadecimal digits, separated with hyphens that demarcate its internal structure and delimited with braces, *{00000000-0000-0000C000-000000000046}*, with the most volatile time elements to the left and the machine identifier to the right.

14. The "ID=" statements in the sessionID_Inbox.xml file tell an XML parser running on the Microsoft Exchange server to replace the respective simple references (i.e., the ID type values "1", "2", "3", "4", and "5") with the associated GUID reference numbers. Each of the "ID=" statements is an indirection to map from a respective ID type value provided by a client to

an associated GUID reference number that is maintained in some session context by the server in the sessionID_Inbox.xml file, where "ID" is a tag that is used to pass the simple reference that replaces the associated GUID reference number.

15. When applied to the exemplary sessionID_Inbox.xml file on page 18, lines 10-34 of the present patent application, the disclosure on page 16, lines 16-19, of the present patent application means that the access module 120 creates the sessionID_Inbox.xml file and passes the simple references (i.e., the ID type values "1", "2", "3", "4", and "5") to the destination device without passing the actual referenced data (i.e., the message items in the "Inbox" folder that are identified by the GUID reference numbers associated with ID type values).

16. I declare that all statements made herein of my own knowledge are true and that all statements made on declaration and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Respectfully submitted,

Date: June 5th 2006

Stanley Foster

Attorney's Docket No.: 10005265-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: MAMOUN ABU-SAMAHA	Art Unit	: 2654
Serial No.	: 09/684,065	Examiner	: Lerner, Martin
Filed	: October 6, 2000		
Title	: DISTRIBUTED VOICE AND WIRELESS INTERFACE MODULES FOR EXPOSING MESSAGING/COLLABORATION DATA TO VOICE AND WIRELESS DEVICES		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

EXHIBIT A

Developing Applications for Microsoft® Exchange with C++

Ben Goetter

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Goetter, Ben, 1964-
Developing Applications for Microsoft Exchange with C++ / Ben Goetter.
p. cm.
Includes index.
ISBN 1-57231-500-8
1. Microsoft Exchange. 2. Client/server computing. 3. C++ (Computer program language) I. Title.
QA76.9.C55G63 1996
005.7'11--dc20
96-28619
CIP

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 MLML 1 0 9 8 7 6

Distributed to the book trade in Canada by Macmillan of Canada, a division of Canada Publishing Corporation.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office. Or contact Microsoft Press International directly at fax (206) 936-7329.

Acquisitions Editor: Casey D. Doyle
Project Editors: Patricia N. Wagner, Caroline Pachaud
Technical Editor: Christina Anagnost

example, *IPersistStorage*) that the interface completely subsumes another interface (in this example, *IPersist*). This is very rare, with one important exception: every interface includes *IUnknown* in such a manner that every interface effectively is *IUnknown*, as if it had inherited *IUnknown*. In other words, every interface delivers the mechanism by which a client can request another interface from the component.

Object Naming

For a client to ask a component object for an interface, the client needs to have a name for that interface: a token that the client and component agree represents the interface and that neither can mistake for any other interface. Recall that any change to an interface results in a new interface distinct from the old one.

Every named object in the universe of COM has as its true name a Globally Unique Identifier, or *GUID*. A GUID is a 128-bit integer, spanning a theoretical range from 0 through $(2^{128}) - 1$, or approximately 3.4×10^{38}.

As the name indicates, each GUID value is "globally unique": it exists to uniquely identify one object class in the universe. A GUID must be unique to be valid; once one has been assigned to name something, it must never be reassigned elsewhere,[4] lest two parties hold two different definitions of the same name. Imagine, for example, what might happen if everybody developed a fondness for the GUID value *1* (certainly much easier to remember and type than a chain of 32 hexadecimal digits or 38 decimal digits). Or picture the confusion that would arise in interface negotiations: a man staggers into a room, sees a figure in a white coat, and gasps, "Are you a doctor?" To which the helpful diesel mechanic responds, "Why, yes I *am*. How may I help you?" Bad news, unless the sick man runs SAE 30W for blood!

Developers allocate GUIDs with an algorithm designed to prevent such collisions. The internal structure of a GUID includes 48 bits of machine-specific unique seed, such as an Ethernet or a Token-Ring network address (if available), representing the system that allocated the GUID, together with 64 bits of time in Universal Time Coordinates (UTC) format, representing the time at which the

4. Fortunately, with 2^{128} possible GUID values, there's no need to allocate them parsimoniously. This range could accommodate assigning a GUID to every hair on every head on every person in the world. Furthermore, in an echo of *Horton Hears a Who*, every hair on our world could actually accommodate an entire tiny world, equally populous and hirsute as our own, and you'd still have enough GUIDs to assign each tiny hair its own value. Indeed, you could repeat this allocation over 200 million Earth-sized planets, each with hairy inhabitants carrying an inhabited world on each hair, before you'd exhaust the space. Aren't orders of magnitude fun?

system allocated it. This allows each of 2.8×10^{14} hosts (the range of the machine-specific identifier) to allocate 10,000,000 GUIDs per second for the next 3000 years (the range of 64-bit UTC).

When a GUID names an interface, the GUID is an interface ID, or *IID*. GUIDs name more than just interfaces, however; everywhere a unique name is necessary, a GUID appears, possibly under a different label. They can name class objects, in which case the GUIDs take the name *CLSIDs*, or they can name remote procedure call (RPC) proxies, in which case they travel as *UUIDs*.[5]

By convention, we write a GUID value as a series of hexadecimal digits, separated with hyphens that demarcate its internal structure and delimited with braces, *{00000000-0000-0000-C000-000000000046}*, with the most volatile time elements to the left and the machine identifier to the right. Because most fleshy, carbon-based people don't work naturally with 128-bit numbers as names, this book uses symbolic constants or shorthand instead of literal GUID values. The quantity listed above, for instance, appears only once in this book and in the API documentation, even though it's the most commonly used interface name in the system: *IUnknown*. Instead of using the literal GUID value, the code refers to the symbol *IID_IUnknown*, while the documents name it as *IUnknown*.

Object Lifespan

Component objects are bodies—logical "objects" in the object-oriented programming sense—that articulate interfaces to their clients. These objects have a life span: they come into existence at a client's demand, fulfill whatever requests are made of them, and finally vanish when they are no longer needed.

The body of code that implements a particular component object is called the *component class*. In the model most common to ActiveX, a client loads the class it needs, passing the system COM library a CLSID—the GUID naming a particular class—and getting in return a class object. A class object (frequently called a *class factory* to reduce overwork of the already grievously abused *o*-word) is itself just a form of component object, one that exports the interface *IClassFactory*; the client interacts with the class factory's offered *IClassFactory* interface to request instances of the component object implemented by the class, as shown in Figure 1-6 on the following page.

5. This is the same UUID that the Open Software Foundation (OSF) Distributed Computing Environment (DCE) Remote Procedure Call mechanism uses.